Enabling multi-view modeling with SysML profiles and model transformations

Aditya A. Shah*, Dirk Schaefer and Christiaan J.J. Paredis

Systems Realization Laboratory,
G.W. WoodruffSchool of Mechanical Engineering,
Georgia Institute of Technology, Atlanta, GA, USA
E-mail: aditya.shah@gatech.edu
E-mail: dirk.schaefer@me.gatech.edu
E-mail: chris.paredis@me.gatech.edu
*Corresponding author

Abstract: Due to increases in system complexity, systems engineering problems often involve many domains, each with their own experts and tools. To help these experts with analysis and decision making, it is desirable to present them with a view of the system that is tailored to their particular task. In this paper, a model integration framework, based on models based systems engineering, is demonstrated to address issues associated with *multi-view modeling*. One important issue discussed in particular is the problem of maintaining consistency between the multiple models and views. The systems modeling language (OMG SysML™) is proposed as a general language to represent the dependencies between the multiple views. Metamodels and graph transformations are defined to map between the views and maintain consistency between them. The integration is achieved in a user-interactive and continuous manner based on declarative transformation rules. The approach is illustrated by applying it to an example problem of an electrical CAD subsystem of a mechatronic system.

Keywords: multi-view modelling; graph transformation; metamodel; systems modeling language; SysML; domain specific language; DSL.

1 Introduction

Systems engineering projects are becoming increasingly complex because they span multiple domains and tools. The increase in complexity has lead to distributed design of systems, both geographically and functionally, consisting of multiple stakeholders with competing objectives. An example of such systems engineering projects are mechatronic systems, which require a combination of mechanical, electrical, electronic and software domains during the design process (Chen et al., 2009). The multi-disciplinary nature of such problems results in large quantities of design data, managed in different tools corresponding to each domain. Maintaining consistency between these multiple data sets and tool-specific models becomes an issue when analyzing different system architectures during the design process. Due to the various subsystems and domains involved in such

systems engineering problems, the ability to describe a system from different viewpoints such as different disciplinary domains, life-cycle phases, or levels of detail, fidelity and abstraction is required. For instance, there could be multiple views associated with a conveyor belt system of a process line for analyzing its performance from different perspectives: an electrical circuit view, a controls view or a simulation view.

A key challenge in dealing with multiple views is that different views of a system all relate to the same system and thus depend on each other. In the conveyor belt system example, simulation and analysis views aid in the specification of system components. However, since different information is represented across multiple views, only portions of different views are related to each other. For instance, component and connection information from schematics are important for simulation views while diagram layout and placement details may not be relevant. Consequently, a mechanism is needed to integrate the required information between views. In current practice, the views are usually represented in different tools and are often maintained independently of each other by the users, resulting in significant non-value-added effort and in significant opportunity for errors. To avoid such costly and risky processes, the approach advocated in model-based systems engineering is to model both the views and the dependencies between them formally in a system model. However, even when using an integrated system model, the modeling formalism must explicitly allow for information to be shared in different views.

In this paper, the focus is therefore on demonstrating a formal method for *multi-view modeling*, i.e., achieving consistency between the *different* information represented through multiple views of a system. For this we explore and develop the foundation for supporting such multiple views in the systems modeling language developed by the object management group (OMG SysML™) (OMG, 2008c). Through domain specific modeling and graph transformations, different domains and the relations between them are described. Moreover, the method presented in this paper complies with industry established standards such as meta object facility (MOF) (OMG, 2006), model driven architecture (MDA) (OMG, 2008b), and SysML, facilitating the integration with a variety of tools and standard frameworks.

In the next section, related work is discussed in more detail. Section 3 describes the use of SysML as a unifying language between multiple views while Section 4 describes the example that will be used to illustrate the method presented in this paper. Section 5 then presents the method that is adopted for multi-view modeling. Section 6 discusses an implementation of the method for integrating an ECAD view and a SysML view of the problem described in Section 4. Finally, Section 7 summarizes the work and outlines future research areas.

2 Related work

The requirements of maintaining consistency between multiple views are different than those for maintaining interoperability between models. Interoperability involves managing similar information across different formats. For instance, standard file formats such as STEP (Lubell¸2002; Chen and Schaefer, 2007) and XML (Czarnecki and Helsen, 2003) are commonly used for interoperability in engineering and software domains. However, multi-view modeling involves managing *different* information across different

tools and domains. In such cases, file formats suffer from certain limitations such as data loss issues in STEP (Alexander et al., 2008) and readability issues with XML.

Since multiple views involve different information, a mechanism to perform operations such as abstraction or addition of information to the model is required. The process of model transformation is suited for this, since it is the automated process of converting a source model into a target model based on specified transformation rules. In these regards, Czarnecki and Helen (2003) discuss the classification and comparison of different model transformation based approaches, such as direct manipulation and graph transformations. In direct manipulation, transformations are defined from scratch using general purpose programming languages such as Java. However, since such languages lack high-level abstractions, the direct manipulation approach can be cumbersome to use, understand and maintain (Sendall and Kozaczynski, 2003). Graph transformations, on the other hand, can be defined in a declarative and visual manner, allowing complex rules to be expressed in an intuitive fashion (Baresi and Heckel, 2002). Moreover, the execution of graph transformations can occur in continuous or incremental modes as opposed to batch mode when using standard file formats. Transformations can therefore be performed interactively with the user which makes managing consistency between views easier (Sendall and Kozaczynski, 2003). In addition, platforms such as MOFLON (2009) provide the capability to define graph transformations and automatically generate corresponding executable code for performing the model transformation in a language such as Java. Therefore, in the research described in this paper, a graph transformation approach is adopted for the integration of multiple views of a system.

In addition to integration between views, a mechanism for systems level design is required for systems that are large and highly complex. For instance, the development of mechatronic systems is highly disjoint due to a lack of integration between different views as well as a lack of systems-level modeling capabilities. To overcome these problems, we propose to use the general purpose modeling language SysML for both systems-level design and for integrating multiple views of the system. The use of SysML as a unifying language for systems design is discussed in the next section.

3 SysML as a unifying language

Multiple views of a system are essential to represent the multi-disciplinary nature of large systems engineering projects such as the design of mechatronic systems. Instead of customized mappings between different views, we believe that a common language such as SysML can serve as a unifying language between the various views of a system. Since each view is mapped to a SysML view, the order of complexity is $0(n)$ as compared to $0(n^2)$ for the case of individual mappings. Moreover, SysML also provides support for systems-level design as well as mechanisms for customizing SysML for specific domains, which are important concepts for MDA as laid out by OMG. Another benefit of SysML is its conformance to industry established standards such as MOF, UML, and XMI, which enables tool support with existing meta-CASE and UML authoring tools.

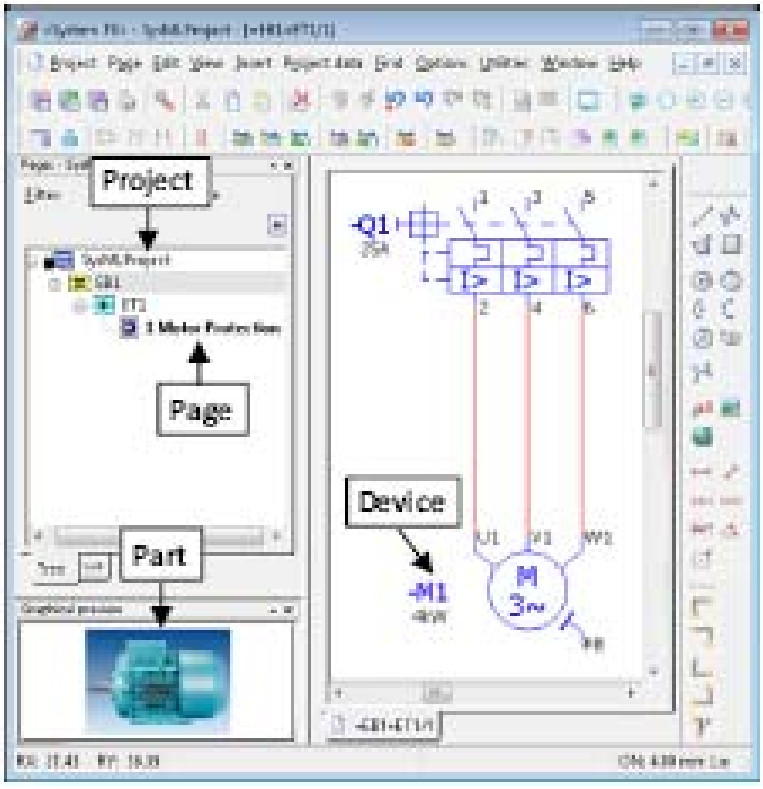
Therefore, we demonstrate a method for the integration between a domain specific view and a SysML view for an example mechatronic system, which is described in the next section.

4 Example problem

A simple mechatronic system with a motor and overload protection switch as one of its subsystems is used to demonstrate the method for multi-view modeling presented in this paper. Electrical systems are designed typically by experts using domain specific tools such as EPLAN Electric P8 (2008). However, since domain specific tools generally lack support for systems-level design as well as integration across multiple domains, the method presented in this paper can be used to achieve integration between multiple views of a system – a system-level view in SysML and a domain-specific view in Electric P8.

In Figure 1, a domain specific view of the example system is shown using a schematic diagram. The system consists of two components: a three-phase motor and a motor overload switch for protection. Constructs such as `Device`, `Part`, `Project` and `Page` are used to model the system. In the next section, a method for multi-view modeling is discussed in the context of the example described above. The application of this method is demonstrated in Section 6, in which a domain-specific view (Figure 1) is automatically generated from a system-level view in SysML (Figure 7).

Figure 1 Domain specific view of motor overload protection subsystem (schematic diagram) in EPLANElectricP8



Source: EPLAN (2008)

5 Multi-view modeling method

Our method for multi-view modeling involves the following steps:

1 formal definition of the domains involved in the system through metamodels

2 customization of SysML through profiles to enable domain specific modeling

3 mapping between the domain specific metamodel and SysML profile through graph transformations

4 tool specific API calls in the transformations to complete the integration between the different views.

The steps outlined above form the foundation for a systematic method for integrating between different views in disparate languages and tools. The method is modularizable, since the same steps are followed to map a domain-specific view with its corresponding SysML view. Thereafter, integrating different tools involves defining transformations within the SysML view of the system.

5.1 Metamodeling

Describing multiple views of the same system in a formal manner is essential for integrating them in a systematic way. Therefore, the first step to integrate multiple views is to define each domain formally by establishing a metamodel. A metamodel defines a language for expressing the information that is relevant in the domain including the relationships between these types of information. For example, the requirements, schematic and simulation views of an ECAD system may be described by different metamodels. They are called *meta*-models because they themselves are models that define the languages in which the models of the views are described. Each view of a system is an instance of a metamodel defined for the domain.

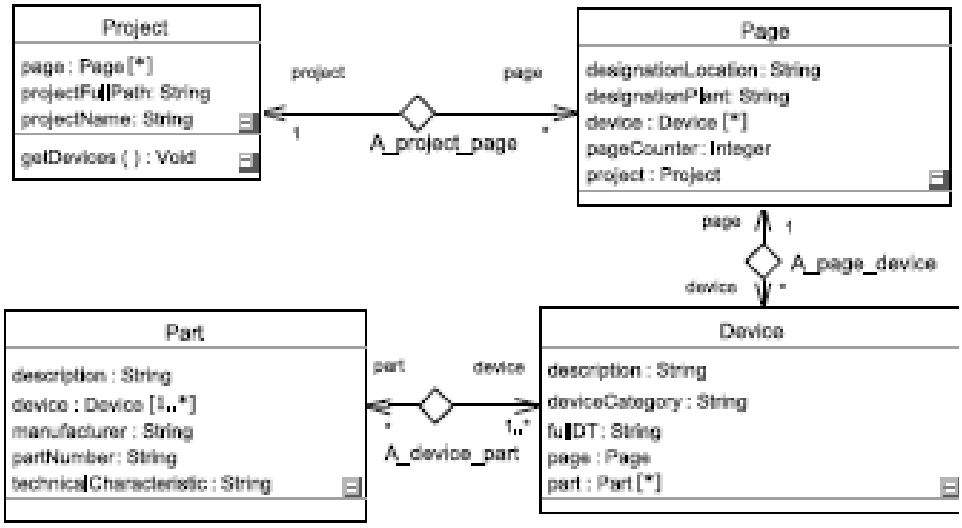
To support model and metamodel driven systems, OMG established the MOF standard. MOF provides a methodology and framework for "defining, manipulating, and integrating meta-data and data in a platform independent manner" (OMG, 2006). The approach we have taken is to specify explicitly the metamodel reconstructed from the API of the tool with which interoperability is desired (Czarnecki, 2005). This is a conversion of the implicit metamodel (i.e., the data structures used internally to the tool and only made visible through its API targeted for general purpose programming languages like C++) into a formal and explicit metamodel compliant with the MOF standard.

The MOF compliant meta-CASE tool MOFLON (2009) is used to define the abstract syntax for the domain specific metamodel for ECAD. A small portion of a metamodel for the ECAD domain is shown in Figure 2. The language constructs associated with the domain are specified as classes (`Project`, `Page`, `Device`, etc.) while the relationships between the various constructs are specified as associations (`A_page_device`, `A_project_page`, etc.). For instance, the metamodel in Figure 2 states that a Project contains a number of `Pages` and can be related to `Pages` through the composition association `A_project_page`.

Once the domain has been formalized by an explicit metamodel, it is necessary to customize SysML to enable domain specific system-level modeling of the ECAD

domain. This is done through the use of SysML profiles, which is the next step in the approach to multi-view modeling.

Figure 2 Portion of metamodel for ECAD domain defining the structure for a valid ECAD system



5.2 Domain specific modeling in SysML

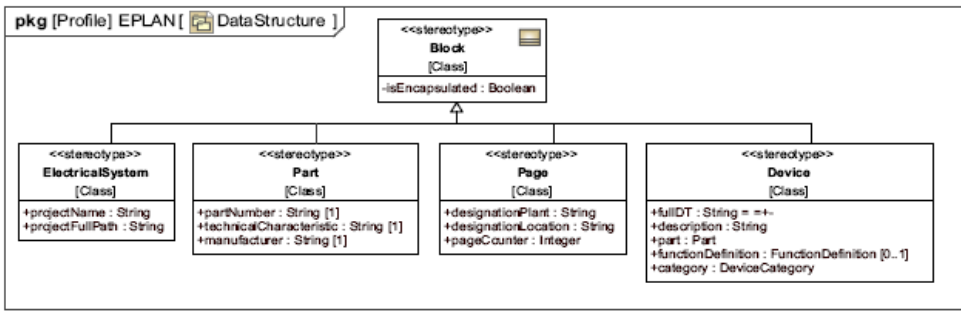
In order for SysML to be a unifying language between multiple views, a mechanism is needed to represent specific domains and systems within SysML. Since SysML is a general purpose modeling language, it lacks the detailed, formal semantics needed for formal domain analysis and automated tool support (Brucker and Doser, 2007). For instance, almost any model can be described using generalized constructs such as SysML blocks. This makes it cumbersome for domain experts to create models in SysML, thereby limiting the acceptance of general SysML for specific domains. To overcome this limitation, SysML provides different ways to create semantics specific to a domain through the use of domain specific languages (DSLs). DSLs simplify commonly used aspects of a domain and reduce the need for repeated use of lower level constructs like value properties needed for their definition. In addition, DSLs enhance computer interpretability since the information in a valid model is encoded at the meta-level instead of the model level. Thus, the use of a DSL makes it easier and more intuitive for system engineers to define ECAD and ECAE designs in SysML. Moreover, DSLs facilitate automated model transformations since mappings are based on language constructs instead of variable names.

Among the various approaches available to define DSLs in SysML, profiles are used since they do not modify the underlying SysML metamodel so that tool support is retained (Weisemoller and Schurr, 2008). A portion of a SysML profile created for the ECAD domain is shown in Figure 3. The profile is constructed as per the domain metamodel and this can be seen from the use of the elements `Page` and `Device` in both the metamodel and profile. Language constructs specific to the ECAD domain (e.g.,

`Page` and `Device`) are defined as stereotypes that extend existing SysML and UML constructs, such as the `Block` metaclass of SysML.

In conclusion, the combination of profiles and metamodels provides the framework in which graph transformations can be applied to integrate these multiple views and maintain consistency among them, which is discussed in the next section on graph transformations.

Figure 3 Portion of SysML profile for ECAD domain used to create structural system-level view, based on the domain metamodel

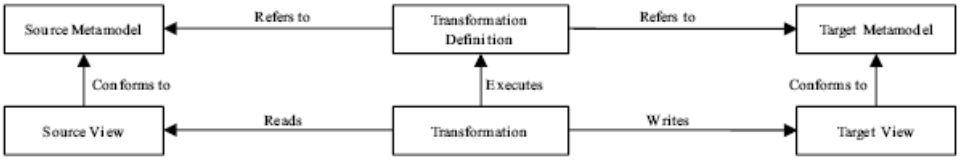


5.3 Graph transformations

Now that we are able to formally define multiple views of the system in different languages, model transformations are used to map between them to provide integration. The domain metamodel and SysML profile can be described in terms of graphs (Baresi and Heckel, 2002), in which the language objects represent the nodes, and associations represent the edges. Consequently, graph transformations can be used to integrate between the different views and models. Two common transformation based approaches are OMG's queries/views/transformations (QVT) (OMG, 2008a) and triple graph grammars (TGG) (Schurr, 1995). A combination of QVT and TGG-like rules are used to define the transformations between the different views. Unlike TGGs, bidirectional transformation rules are not automatically generated and therefore mappings are required in each direction. For instance, the example mapping described in this section is from SysML to EPLAN. Another mapping would be defined from EPLAN to SysML.

The process of maintaining consistency between multiple views involves creation of new or updated views through the execution of graph transformations. This process is shown in Figure 4, in which transformations are defined at the meta-level on the corresponding metamodels and not the models themselves.

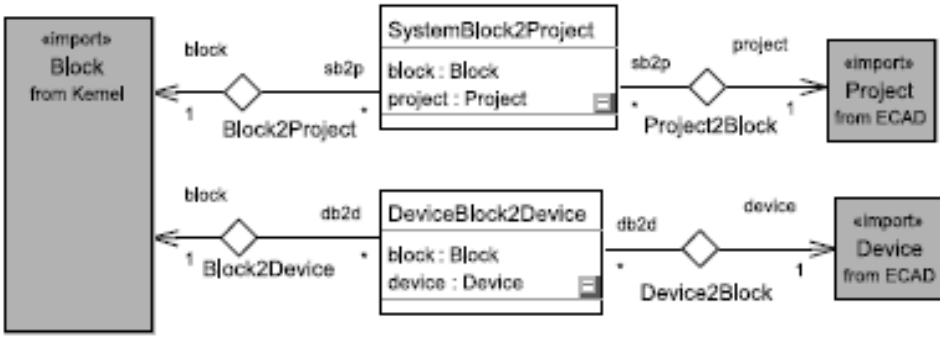
Figure 4 Process of model transformation: from source to target model



Source: Czarnecki and Helsen (2006)

A correspondence metamodel is used to link elements of the source and target metamodels on which the transformations are defined (Schurr, 1995). The correspondence metamodel for the ECAD system, shown in Figure 5, links objects of the SysML view to the ECAD tool specific view. For instance, the object `DeviceBlock2Device` links a `Block` object in SysML to a `Device` object in the ECAD domain, just as the stereotype `Device` extends from the `Block` class in the SysML profile.

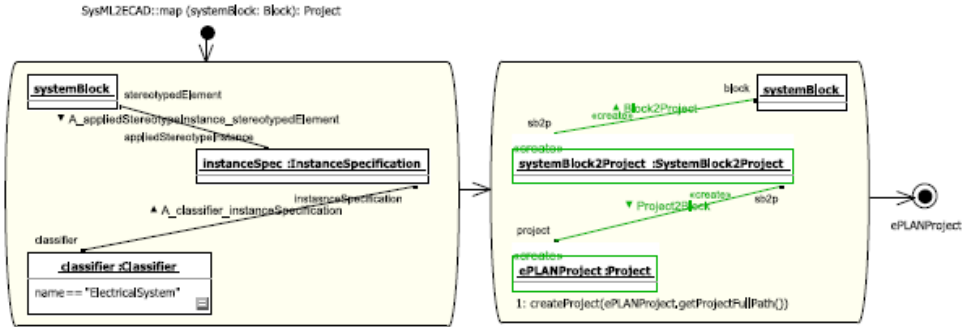
Figure 5 Correspondence metamodel that links objects of a SysML view with objects of an ECAD view



After formally defining the language of the views through metamodeling, profiles, and a correspondence metamodel, it is now possible to transform from one view into another through the definition and subsequent execution of graph transformations. The graph transformations are written in a declarative and graphical manner through the use of story diagrams (Fischer et al., 2000). In Figure 6, a story diagram is shown in which a block of stereotype `ElectricalSystem` in SysML is transformed into a `Project` file in the ECAD-specific tool EPLAN. The input to the story diagram is a SysML block and the output is an EPLAN-specific project file. MOFLON is used to automatically generate Java metadata interface (JMI) compliant Java code, which allows for easier integration and execution of transformations from within standard SysML tools.

Thus, through the successive execution of graph transformations, it is possible to generate the target view (in EPLAN) from the source view (in SysML), thereby automating the process of maintaining consistency between different views.

Figure 6 Story diagram that transforms electrical system block in SysML to a project in EPLAN (ECAD specific tool)



6 SysML and ECAD integration

The method presented in the previous section is applied to the example mechatronic system described in Section 4. Figure 7 shows a systems-level structural view in SysML, which is modeled using the domain-specific constructs specified in the SysML profile. The constructs defined in the profile are applied as stereotypes onto generic SysML constructs such as blocks. This automatically assigns properties to the blocks, which simplifies the task of the modeler.

Graph transformations are then executed, in the form of a Java plugin, on the SysML model to derive an intermediate domain model of the system as per the metamodel that is defined. This domain model is then converted to the domain-specific view (EPLAN, for this example) through the use of function calls to the internal API of the tool (EPLAN). JNBridgePro (JNBridge, 2008) is used to handle the incompatibility between the languages of the SysML tool (Java) and that of EPLAN (.NET).

In Figure 8, the complete process is shown of creating the SysML view, applying the graph transformation to create the domain model and then using API calls to create the tool specific view. The resultant EPLAN view is shown in Figure 1, which consists of two devices as defined in the SysML view but with added information as compared to the SysML view, such as visual representation of components through the use of symbols.

Figure 7 Structural view of the system in SysML – a block definition diagram (BDD) created using the constructs specified in the SysML Profile for the ECAD domain (Figure 3)

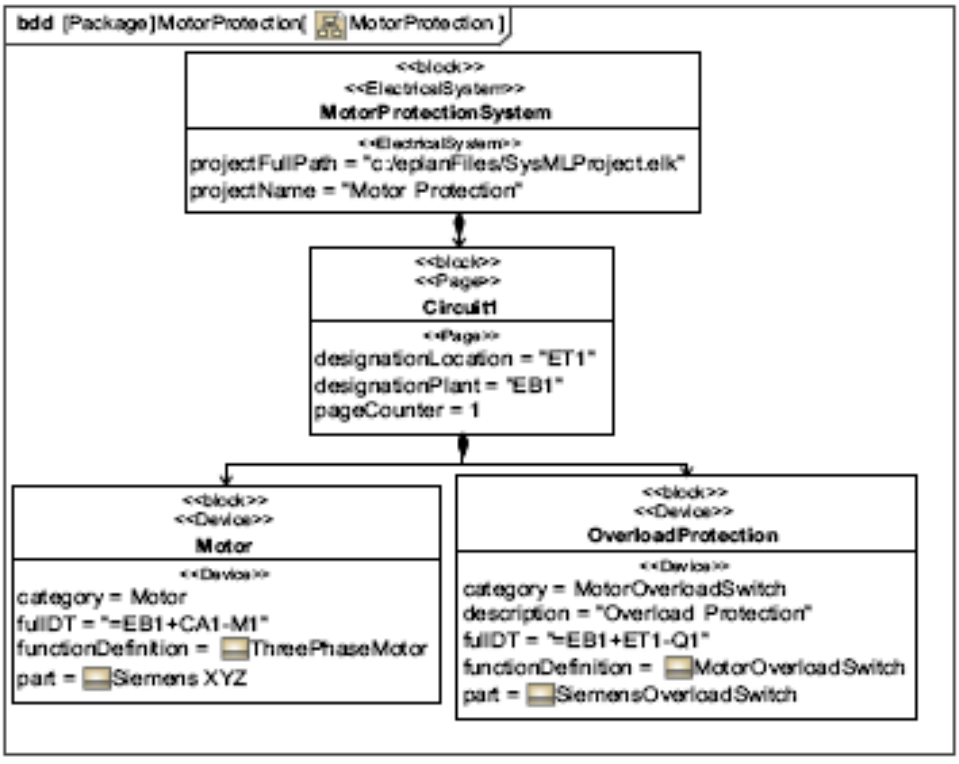
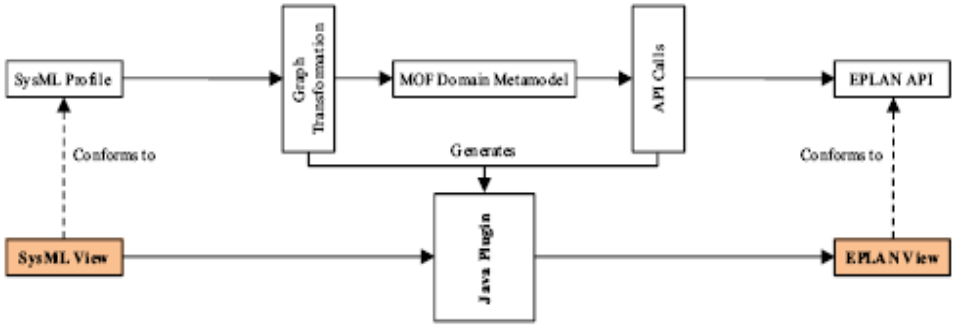


Figure 8 Process for implementing the multi-view modeling method: from SysML view to EPLAN view through execution of graph transformations (Java plugin)



7 Closure and future work

In this paper, the use of SysML profiles and graph transformations for multi-view modeling is discussed. The method presented in this paper integrates multiple views that contain different information, thereby leveraging the capabilities of the different languages and domains involved. For instance, electrical CAE tools such as EPLAN are well-suited for detailed design of power and control systems through diagrams such as schematics or wiring diagrams. SysML, on the other hand, is better suited for defining the high-level relationships that exist between requirements, structure, and behavior. Consequently, this method provides the designer with the ability to trace decisions made to corresponding requirements defined in SysML as well as maintain bidirectional consistency between other domains that are linked with SysML. This is different from current approaches for interoperability, which export the complete model into standard file formats to achieve single-direction integration.

The general-purpose and customizable nature of SysML provides a good opportunity for scalable information exchange between domains. The formal description of domains, through metamodels and SysML profiles, and the creation of mappings through story diagrams allow designers to start from a systems perspective and automatically generate domain specific models that are necessary for the latter stages of the design process. Moreover, the use of domain models in the abstract syntax provides a level of tool independence since different tools of the same domain do not require new metamodels, only different API calls. Thus, the method outlined in this paper is a step towards the unification of the various domains involved in the solving of multi-disciplinary systems engineering problems.

Future work in this area involves refining the process of integrating tool specific APIs with graph transformations through the use of JMI to enable the use of TGG for bidirectional mapping between multiple views of the system.

Acknowledgements

The authors would like to thank Julie Bankston, Roger Burkhart, Sanford Friedenthal, Aleksandr Kerzhner, Leon McGinnis, and Russell Peak for the discussions that helped crystallize the ideas presented in this paper. No Magic Inc. provided access to its MagicDraw UML/SysML tool, EPLAN Software and Services LLC provided access to Electric P8, and JNBridge LLC. provided access to JNBridgePro.

References

Alexander, B., Lian, D. and Manjula, P. (2008) 'An approach to accessing product data across system and software revisions', *Advanced Engineering Informatics*, Vol. 22, No. 2, pp.222–235.

Baresi, L. and Heckel, R. (2002) 'Tutorial introduction to graph transformation: a software engineering perspective', *Graph Transformation*, pp.402–429.

Brucker, A.D. and Doser, J. (2007) 'Metamodel-based UML notations for domain-specific languages', in 4th *International Workshop on Software Language Engineering (ATEM2007)*, Nashville, USA.

Chen, K. and Schaefer, D. (2007) 'MCAD – ECAD integration: overview and future research perspectives', *Proceedings of the 2007 ASME International Mechanical Engineering Congress and Exposition*, Seattle, Washington, USA.

Chen, K., Bankston, J., Panchal, J.H. and Schaefer, D. (2009) 'A framework for integrated design of mechatronic systems', *Collaborative Design and Planning for Digital Manufacturing*, pp.37–70.

Czarnecki, K. (2005) 'Overview of generative software development', *Unconventional Programming Paradigms*, pp.326–34l.

Czarnecki, K. and Helsen, S. (2003) 'Classification of model transformation approaches', *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*.

Czarnecki, K. and Helsen, S. (2006), 'Feature-based survey of model transformation approaches', *IBM Systems Journal*, Vol. 45, No. 3, pp.62l–645.

EPLAN, S.S.L. (2008) 'Eplan Electric P8', available at http://www.eplanusa.com/.

Fischer, T., Niere, J., Torunski, L. and Zundorf, A. (2000) 'Story diagrams: a new graph rewrite language based on the unified modeling language and Java', *Theory and Application of Graph Transformations*, pp.l57–l67.

JNBridge, L. (2008) 'Jnbridgepro', available at http://www.jnbridge.com/jnbpro.htm.

Lubell, J. (2002) 'From model to markup: XML representation of product data', *XML Conference*, pp.8–l3, Baltimore, MD.

MOFLON (2009) 'MOFLON homepage', available at http://moflon.org/.

OMG (2006) 'Meta object facility (MOF) core specification V2.0', available at http://www.omg.org/docs/formal/06-0l-0l.pdf.

OMG (2008a) 'Meta object facility (MOF) 2.0 query/view/transformation, V1.0', available at http://www.omg.org/docs/formal/08-04-03.pdf.

OMG (2008b) 'OMG model driven architecture', available at http://www.omg.org/mda/.

OMG (2008c) 'OMG systems modeling language, V1.1', available at http://www.omg.org/docs/formal/08-11-02.pdf.

Schurr, A. (l995) 'Specification of graph translators with triple graph grammars', *Graph-Theoretic Concepts in Computer Science*, pp.l5l–l63.

Sendall, S. and Kozaczynski, W. (2003) 'Model transformation: the heart and soul of model-driven software development', *Software, IEEE*, Vol. 20, No. 5, pp.42–45.

Weisemoller, I. and Schurr, A. (2008) 'A comparison of standard compliant ways to define domain specific languages', *Models in Software Engineering*, pp.47–58.